Propositional logic

Formulas Φ of propositional logic are given by the rules below:

- If p is a propositional variable then p is a formula
- if Φ and Ψ are formulas, then so are $\Phi \wedge \Psi$ conjunction, $\Phi \vee \Psi$ disjunction, $\Phi \to \Psi$ implication, $\neg\Phi$ negation
- $\bot, \top$ false, true (truth constants)

EXAMPLE: $p \wedge (q \vee r)$

We can define formulas using a grammar $\Phi ::= p \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \to \Phi \mid \neg\Phi \mid \top \mid \bot$

Proof rules allow us to prove tautologies (logically valid formulas)

NATURAL DEDUCTION RULES FOR PROPOSITIONAL LOGIC

CONJUNCTION

introduction:

$$\frac{\Phi \quad \Psi}{\Phi \wedge \Psi}\ (\wedge i)$$

elimination:

$$\frac{\Phi \wedge \Psi}{\Phi}\ (\wedge e_1) \qquad \frac{\Phi \wedge \Psi}{\Psi}\ (\wedge e_2)$$

DISJUNCTION

$$\frac{\Phi}{\Phi \vee \Psi}\ (\vee i_1) \qquad \frac{\Psi}{\Phi \vee \Psi}\ (\vee i_2)$$

$$\frac{\Phi \vee \Psi \qquad \boxed{\begin{matrix}\Phi \\ \vdots \\ \chi\end{matrix}} \qquad \boxed{\begin{matrix}\Psi \\ \vdots \\ \chi\end{matrix}}}{\chi}\ (\vee e)$$

IMPLICATION

$$\frac{\boxed{\begin{matrix}\Phi \\ \vdots \\ \Psi\end{matrix}}}{\Phi \to \Psi}\ (\to i)$$

box = no longer available, we discharge the assumption Φ

$$\frac{\Phi \to \Psi \quad \Phi}{\Psi}\ (\to e)$$ Modus ponens

NEGATION

$$\frac{\boxed{\begin{matrix}\Phi \\ \vdots \\ \bot\end{matrix}}}{\neg\Phi}\ (\neg i)$$

$$\frac{\neg\Phi \quad \Phi}{\bot}\ (\neg e)$$

FALSITY

$$\frac{\bot}{\Phi}\ (\bot e)$$

$$\frac{}{\top}\ (\top i)$$

REDUCTIO AD ABSURDUM (gives classical logic)

$$\frac{\boxed{\begin{matrix}\neg\Phi \\ \vdots \\ \bot\end{matrix}}}{\Phi}\ (raa)$$

raa = pcb = proof by contradiction

If we add raa, we can prove LAW OF EXCLUDED MIDDLE $\Phi \vee \neg\Phi$

Eg. $p \wedge q \to q \wedge p$

1.	$p \wedge q$	assumption
2.	p	$\wedge e_1$ on 1.
3.	q	$\wedge e_2$ on 1.

Eg. $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$

$\longrightarrow$

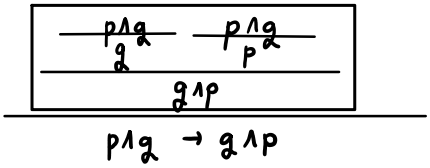
1.	$p \wedge (q \vee r)$	assumption
2.	p	$\wedge e_1$ on 1.

$\longleftarrow$ HW

3.	q	∧e₂ on 1.
4.	q∧p	∧i on 2.,3.

5. p∧q → q∧p →i

PROOF AS A TREE:



2.	p	∧e₁ on 1.
3.	q∨r	∧e₂ on 1.
4.	q	assumption
5.	p∧q	∧i on 2.,4.
6.	(p∧q)∨(p∧r)	∨i₁ on 5.
7.	r	assumption
8.	p∧r	∧i on 2.,6.
9.	(p∧q)∨(p∧r)	∨i₂ on 8.
10.	(p∧q)∨(p∧r)	∨e on 3.,4.-6.,7.-9.

11. p∧(q∨r) → (p∧q)∨(p∧r) →i on 1.-10.

Eg. Φ∨¬Φ

1.	¬(Φ∨¬Φ)	assumption
2.	Φ	assumption
3.	Φ∨¬Φ	∨i₂ on 2.
4.	⊥	¬e on 1.,3.
5.	¬Φ	¬i on 2.-4.
6.	Φ∨¬Φ	∨i₂ on 5.
7.	⊥	¬e on 1.,6.
8.	Φ∨¬Φ	pbc on 1.-7.

SIMPLE TYPES (with sums and products)

$A ::= \alpha \mid 0 \mid A \times A \mid A + A \mid A \to A$

PROOF RULES

PRODUCT

$$\frac{s:A \quad t:B}{(s,t):A\times B}\ (\times i)$$

$$\frac{t:A\times B}{pr_1 t:A}\ (\times e_1)$$

$$\frac{t:A\times B}{pr_2 t:B}\ (\times e_2)$$

SUM

$$\frac{t:A}{in_1(t):A+B}\ (+i_1)$$

$$\frac{t:B}{in_2(t):A+B}\ (+i_2)$$

$$\frac{s:A+B \quad \boxed{\begin{matrix}x:A\\ \vdots\\ t:C\end{matrix}} \quad \boxed{\begin{matrix}y:B\\ \vdots\\ u:C\end{matrix}}}{\text{case } d\,(in_1(x:A)\mapsto t \mid in_2(y:B)\mapsto u) : C}\ (+e)$$

EMPTY TYPE

$$\frac{t:0}{\text{empty}(t):A}\ (0e)$$

FUNCTION

$$\frac{\boxed{\begin{matrix}x:A\\ \vdots\\ t:B\end{matrix}}}{\lambda x:A.\ t : A\to B}\ (\to i)$$

$$\frac{s:A\to B \quad t:A}{st:B}\ (\to e)$$

COMPUTATION RULES

$$\mathrm{pr}_1(s,t) \to s$$

$$\mathrm{pr}_2(s,t) \to t$$

$$\mathrm{case}(\mathrm{in}_1(s))\ (\mathrm{in}_1(x{:}A) \mapsto t \mid \mathrm{in}_2(y{:}B) \mapsto u) \to t[s/x]$$

$$\mathrm{case}(\mathrm{in}_2(s))\ (\mathrm{in}_1(x{:}A) \mapsto t \mid \mathrm{in}_2(y{:}B) \mapsto u) \to u[s/y]$$

$$(\lambda x{:}A.\ s)t \to s[t/x]$$

Deriving a term of type $A \times B \to B \times A$:

$$\dfrac{\dfrac{x: A\times B}{\mathrm{pr}_2(x): B} \quad \dfrac{x: A\times B}{\mathrm{pr}_1(x): A}}{(\mathrm{pr}_2(x), \mathrm{pr}_1(x)) : B\times A}$$
$$\lambda x{:}A\times B.\ (\mathrm{pr}_2(x), \mathrm{pr}_1(x)) : A\times B \to B\times A$$

We use previous derivation of distributivity to derive a term $A\times(B+C) \to (A\times B)+(A\times C)$

$F := \lambda x{:}A\times(B+C).\ \mathrm{case}(\mathrm{pr}_2(x))\left[\mathrm{in}_1(y{:}B) \mapsto \mathrm{in}_1(\mathrm{pr}_1(x), y) \mid \mathrm{in}_2(z{:}C) \mapsto \mathrm{in}_2(\mathrm{pr}_1(x), z)\right] : A\times(B+C) \to (A\times B)+(A\times C)$

$x: A\times(B+C)$ assumption

$\mathrm{pr}_1(x): A$

$\mathrm{pr}_2(x): B+C$

$y: B$ assumption

$(\mathrm{pr}_1(x), y): A\times B$

$\mathrm{in}_1(\mathrm{pr}_1(x), y) : (A\times B)+(A\times C)$

$z: C$ assumption

$(\mathrm{pr}_1(x), z) : A\times C$

$\mathrm{in}_2(\mathrm{pr}_1(x), z): (A\times B)+(A\times C)$

$\mathrm{case}(\mathrm{pr}_2(x))(\ldots) : (A\times B)+(A\times C)$

$\lambda x{:}A\times(B+C).\ \mathrm{case}(\mathrm{pr}_2(x))(\ldots) : A\times(B+C) \to (A\times B)+(A\times C)$

$A = \mathrm{Nat},\ B = \mathrm{list\ nat},\ C = \mathrm{bool}$

$F(17, \mathrm{in}_2(\mathrm{false})) \to \mathrm{case}(\mathrm{pr}_2(17, \mathrm{in}_2(\mathrm{false})))\ [\ldots]$

$\to \mathrm{case}(\mathrm{in}_2(\mathrm{false}))\ [\ldots]$

$\to \ldots$

$\to \mathrm{in}_2(17, \mathrm{false})$

Predicate logic

Predicate logic (first order logic) extends propositional logic with quantifiers ($\forall, \exists$)

THE STANDARD APPROACH TO PREDICATE LOGIC (which we won't follow)

Choose a "universe of discourse" e.g. $\mathbb{N}$, $\mathbb{R}$, groups, rings ...
Give yourself some **terms** for expressing elements of the universe.
e.g. terms for the case of $\mathbb{N}$ or $\mathbb{R}$: $t ::= x \mid 0 \mid t+t \mid t*t \mid 1$
or for a group $t ::= x \mid e \mid t \cdot t \mid$ we can define t^{-1} by logic

Give yourself some **predicates** expressing properties of the universe.
E.g. in the case of $\mathbb{N}$ or $\mathbb{R}$, we might give ourselves $<$ as a binary predicate.
Predicates express relations.

↓
this gives us atomic formulas of the form $t_1 < t_2$, $t_1 = t_2$

The grammar for formulas extends propositional logic by:

$$\phi ::= P(t_1, \dots, t_k) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \to \phi \mid \bot \mid \forall x.\phi \mid \exists x.\phi$$

predicate taking k arguments

omic formula

at

Examples of formulas of first-order logic using terms formed with $\cdot, 1, 0$ and the $<$ relation:

$$\forall x.\ (\underbrace{0 < x}_{\text{atomic}} \to \exists y.\ \underbrace{x = y \cdot y}_{\text{atomic f.}})$$

Every variable is **bound**. The formula has no free variables.
The formula is false in $\mathbb{N}$ and true in $\mathbb{R}$.

The major limitation of traditional first-order logic is the restriction to a single universe (for variables and quantifiers).

If one wants to reason about manipulating lists of nat. numbers one cannot naturally incorporate relations such as
$n \in \ell$ the number n belongs to list ℓ.

We address this by unifying logic and type theory. Specifically we build logic on top of type theory. To do this we replace the grammar for formulas with a system of rules for generating formulas.

Type theory gives a system of rules for manipulating judgements $t : A$ "term t has type A"
Propositional logic ϕ "ϕ holds"
Combine the two using additional rules $\phi : \mathrm{Prop}$ "ϕ is a proposition"

Replace grammar $\phi ::= p \mid \bot \mid T \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \to \phi$ with rules for a **UNIVERSE Prop OF PROPOSITIONS**:

Propositional logic: $\dfrac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \wedge \Psi : \text{Prop}}$ $\qquad$ $\dfrac{}{\bot : \text{Prop}}$ $\qquad$ $\dfrac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \vee \Psi : \text{Prop}}$ $\qquad$ $\dfrac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \to \Psi : \text{Prop}}$

Adding quantifiers: $\dfrac{\begin{array}{|c|}\hline x:A \\ \vdots \\ \Phi : \text{Prop} \\ \hline \end{array}}{(\forall x:A.\ \Phi) : \text{Prop}}$ $\qquad$ $\dfrac{\begin{array}{|c|}\hline x:A \\ \vdots \\ \Phi : \text{Prop} \\ \hline \end{array}}{(\exists x:A.\ \Phi) : \text{Prop}}$

Suppose we have a constant in-list: nat → nat list → Prop

in-list n ℓ is the proposition stating that n is contained in the list ℓ.

Then we derive for example: $(\forall x:\text{nat list}.\ \exists y:\text{nat}.\ \text{n-list}\ x\ y) : \text{Prop}$

PROOF RULES FOR QUANTIFIERS

∀

$$\dfrac{\begin{array}{|c|}\hline x:A \\ \vdots \\ \Phi \\ \hline \end{array} \qquad \begin{array}{|c|}\hline x:A \\ \vdots \\ \Phi : \text{Prop} \\ \hline \end{array}^{*} \qquad A : \text{Type}^{*}}{\forall x:A.\ \Phi}\ \forall i$$

* important: the free variable x is not used inside the box.

$$\dfrac{\forall x:A.\ \Phi \qquad t:A \qquad ^{*}\ ^{*}}{\Phi[t/x]}\ \forall e$$

$G[t/x]$:= take G, rename all bound variables in G to be different from x and from all variables in t. and substitute the term t for all (free) occurances of x in G.

From now on we identify expression up to α-equivalence (up to renaming of bound variables) e.g. $\forall x:A.Px$ is equal to $\forall y:A.Py$.

Suppose we have $P : A \to \text{Prop}$, $Q : A \to \text{Prop}$ where A is a type.

We prove $(\forall x:A.\ Px) \wedge (\forall x:A.\ Qx) \longrightarrow \forall x:A.\ (Px \wedge Qx)$

1.	$(\forall x:A.\ Px) \wedge (\forall x:A.\ Qx)$	assumption
2.	$\forall x:A.\ Px$	$\wedge e_1$ on 1.
3.	$\forall x:A.\ Qx$	$\wedge e_2$ on 1.
4.	$y:A$	assumption
5.	Py	$\forall e$ on 2., 4.
6.	Qy	$\forall e$ on 3., 4.
7.	$Py \wedge Qy$	$\wedge i$ on 5, 6
8.	$\forall x:A.\ (Px \wedge Qx)$	$\forall i$ on 4.-7.
9.	conclusion	$\to i$ on 1.- 8.

∃

$$\dfrac{\Phi[t/x] \qquad t:A}{\exists x:A.\ \Phi}\ \exists i$$

$$\dfrac{\exists x:A.\ \Phi \qquad \begin{array}{|c|}\hline x:A \\ \Phi \\ \vdots \\ \Psi \\ \hline \end{array} \qquad \begin{array}{|c|}\hline x:A \\ \vdots \\ \Phi : \text{Prop} \\ \hline \end{array} \qquad \Psi : \text{Prop}}{\Psi}\ \exists e$$

Homework: Suppose $P : A \to \text{Prop}$, $Q : \text{Prop}$. Prove $(\exists x:A.\ Px) \wedge Q \longrightarrow \exists x:A.\ (Px \wedge Q)$ also that ∃ distributes over disjunction.

Given p : Prop and $q : A \to$ Prop, a proof of $p \wedge (\exists x : A.\, q\, x) \to \exists x : A.\, p \wedge q\, x$

1	$p \wedge \exists x : A.\, q\, x$	assumption
2	p	$\wedge e$ on 1
3	$\exists x : A.\, q\, x$	$\wedge e$ on 1
4	$y : A$	assumption
5	$q\, y$	assumption
7	$p \wedge q\, y$	$\wedge i$ on 2,5
8	$\exists x : A.\, p \wedge q\, x$	$\exists i$ on 7
9	$\exists x : A.\, p \wedge q\, x$	$\exists e$ on 3, 4–8
10	$p \wedge (\exists x : A.\, q\, x) \to \exists x : A.\, p \wedge q\, x$	$\to i$ on 1–9

Add a new judgement form A : Type "A is a type"

Replace the grammar $A ::= 0 \mid 1 \mid A \times A \mid A + A \mid A \to A$ with rules for a **UNIVERSE Type OF TYPES**:

$$\frac{}{0 : \text{Type}} \qquad \frac{A : \text{Type} \quad B : \text{Type}}{A \times B \ \text{Type}} \qquad \frac{A : \text{Type} \quad B : \text{Type}}{A + B : \text{Type}} \qquad \frac{A : \text{Type} \quad B : \text{Type}}{A \to B : \text{Type}}$$

Distinguish beetwen the judgements: $A : \text{Type}$ "A is a well-formed type"

$t : A$ "t is an element of type A"

DEPENDENT PRODUCTS (Π TYPES)

In maths, we have a general product construction $\prod_{i \in I} A_i$ ... the product of a family $(A_i)_{i \in I}$ of sets. As a type $\prod_{x:A} B$. A is a type and B is a type depending on x.

Well-formedness rules:

$$\frac{\begin{array}{c}[x : A] \\ \vdots \\ B : \text{Type}\end{array}}{(\Pi_{x:A}.\ B) : \text{Type}}$$

Introduction and elimination rules:

$$\frac{\begin{array}{c}[x : A] \\ \vdots \\ t : B\end{array}}{\lambda x : A.\, t \ : \ \Pi_{x:A} B}\ \Pi i \qquad \frac{s : \Pi_{x:A} B \quad t : A}{s\,t \ : \ B[t/x]}$$

$A \to B$ is defined in Lean as in abbreviation for $\Pi_{x:A} B$ where B does not depent on x.

$\forall x : A.\ \phi$ is also a special case of dependent product.

Interesting types

Adding new types (or type constructors):

- formation rules (for showing $A : \text{Type}$, where A is the new construction)
- introduction rules (for creating terms inhabiting the type)
- elimination rules (how we use terms inhabiting the type)
- computation rules (how we compute with terms inhabiting the type)

NATURAL NUMBERS AS A TYPE

Formation rule: $\dfrac{}{\text{nat} : \text{Type}}$

Introduction rules: $\dfrac{}{0 : \text{nat}}$ $\qquad \dfrac{n : \text{nat}}{\text{succ } n : \text{nat}}$ or equivalently, consider succesor as a constant : $\text{succ} : \text{nat} \to \text{nat}$

TYPES OF LISTS

Formation rule: $\dfrac{A : \text{Type}}{\text{list } A : \text{Type}}$ or a constant $\text{list} : \text{Type} \to \text{Type}$

Introduction rules: $\dfrac{A : \text{Type}}{\text{nil}_A : \text{list } A}$ $\qquad \dfrac{a : A \quad as : \text{list } A}{\text{cons } a \; as : \text{list } A}$ or $\text{cons} : A \to \text{list } A \to \text{list } A$

TYPES OF VECTORS

(a, b) of length 2 where $a : A$, $b : A$
$\text{cons } a \; 1 \; (\text{cons } b \; 0 \; \text{nil}) : \text{vector } A \; 2$

Formation rule: $\dfrac{A : \text{Type} \quad n : \text{nat}}{\text{vector } A \; n : \text{Type}}$ or $\text{Type} \to \text{nat} \to \text{Type}$

Introduction rules: $\dfrac{A : \text{Type}}{\text{nil} : \text{Vector } A \; 0}$ $\qquad \dfrac{a : A \quad as : \text{vector } A \; n}{\text{cons } a \; n \; as : \text{Vector } A \; (\text{suc } n)}$

or $\text{cons} : A \to \Pi n : \text{nat}. \; (\text{vector } A \; n \longrightarrow \text{vector } A \; (\text{succ } n))$

DEPENDENT PRODUCTS AND SUMS (Π AND Σ TYPES)

Formation rule: $\dfrac{\begin{array}{|c|}\hline x : A \\ \vdots \\ B : \text{Type} \\ \hline \end{array}}{\Pi x : A. B : \text{Type}}$

Introduction rule: $\dfrac{\begin{array}{|c|}\hline x : A \\ \vdots \\ t : B \\ \hline \end{array}}{\lambda x. A : t : \Pi x : A \; B}$

Elimination rule: $\dfrac{s : \Pi x : A. B \qquad t : A}{s\,t \; : \; B[t/x]}$

Computation rule: $(\lambda x : A. s)\, t \longrightarrow s[t/x]$

$A \to B$ is defined to be $\Pi x : A. B$ where $x \notin Fv(B)$

Formation rule: $\dfrac{\begin{array}{|c|}\hline x : A \\ \vdots \\ B : \text{Type} \\ \hline \end{array}}{\Sigma x : A. B : \text{Type}}$

Introduction: $\dfrac{s : A \qquad t : B[s/x]}{(s, t) : \Sigma x : A. B}$

Elimination: $\dfrac{s : \Sigma x : A. B}{\text{pr}_1(s) : A}$ $\qquad \dfrac{s : \Sigma x : A. B}{\text{pr}_2(s) : B[\text{pr}_1(s)/x]}$

Computation: $\text{pr}_1(s, t) \longrightarrow s$
$\text{pr}_2(s, t) \longrightarrow t$

One can define $A \times B$ as $\Sigma x : A. B$ where $x \notin Fv(B)$.

TYPES AND THEIR ELEMENTS

0 ... no elts
$A \times B$ ... pairs (a,b) where $a:A$ and $b:B$
$A + B$ ... $in_1(a)$ and $in_2(b)$ where -"-
$A \to B$ ... functions f that map $a:A$ to $fa:B$
$\Pi x:A.\ B$ ... functions f that map $a:A$ to $fa: B[a/x]$
$\Sigma x:A.\ B$ ... pairs (a,b) where $a:A$ and $b: B[a/x]$.

THE BROUWER-HEYTING-KOLMOGOROV INTERPRETATION OF LOGIC (BHK)

proofs of $\bot$ ... there is no proof of false
$\Phi \wedge \Psi$ ... pair (a,b) where a proves Φ and b proves Ψ
$\Phi \vee \Psi$ ... either $in_1(a)$ where a proves Φ or $in_2(b)$ where b proves Ψ
$\Phi \to \Psi$ ... a function f that maps proofs a of Φ to proofs fa of Ψ.
$\forall x:A.\ \Phi$ ... a function f that maps elements $a:A$ to proofs fa of $\Phi[a/x]$.
$\exists x:A.\ \Phi$ ... a pair (a,b) where $a:A$ and b proves that $\Phi[a/x]$.

THE CURRY-HOWARD CORRESPONDENCE: Propositions as types

If we have dependent type theory, we don't need to consider logic separately, instead we use the type constructs for the logical connections and quantifiers.

Instead of	we use
$\bot$	0
$\Phi \wedge \Psi$	$A \times B$
$\Phi \vee \Psi$	$A + B$
$\Phi \to \Psi$	$A \to B$
$\forall x:A.\ \Phi$	$\Pi x:A.\ B$
$\exists x:A.\ \Phi$	$\Sigma x:A.\ B$

This approach was developed by Martin-Löf 1970s with the development of dependent type theory (MLTT).
View propositions as types whose elements are proofs of the proposition.

PROPOSITIONS AS SUBTERMINAL TYPES (the Lean approach to logic)

Lean takes a different approach: propositions as subterminal types.
If $\Phi: Prop$ then $\Phi: Type$ (Φ is the type of all proofs of the proposition Φ)
Subterminal property: if $a:\Phi$ and $b:\Phi$ then $a=b$.
We also need $Prop: Type$. (Prop is a type)

THE SCOTT-PRAWITZ INTERPRETATION OF LOGIC

* type-theoretic definition of connectives and quantifiers

$\bot$	$\Pi p: prop.\ p$
$\Phi \wedge \Psi$	$\Pi p: prop.\ (\Phi \to (\Psi \to P)) \to P$
$\Phi \vee \Psi$	$\Pi p: prop.\ (\Phi \to P) \to (\Psi \to P) \to P$
$\Phi \to \Psi$	$\Phi \to \Psi$
$\forall x:A.\ \Phi$	$\Pi x:A.\ \Phi$
$\exists x:A.\ \Phi$	$\Pi p:prop.\ (\Pi x:A.\ (\Phi \to P)) \to P$

The logical connectives and quantifiers are all defined in terms of the Π type constructor alone (since $\to$ is a special case of Π)

$$\frac{\Phi \quad \Psi}{\Phi \wedge \Psi}$$

suppose we have $s: \Phi$ $t: \Psi$
then $\lambda p: prop.\ \lambda f: \Phi \to \Psi \to P\ \ f\ s\ t$

$$\frac{\Phi \wedge \Psi}{\Psi}$$

Suppose $u: \Phi \wedge \Psi$ ie.
$u: \Pi p:prop\ (\Phi \to \Psi \to P) \to P$
then $u\ \Psi\ (\lambda a:\Phi.\ \lambda b.\Psi.\ b) : \Psi$

ELIMINATION AND COMPUTATION RULES FOR nat (Principal of induction for natural numbers)

The eliminator for nat (R - recurser)

$R_{nat}: \Pi p{:}nat \to Type.\ P0 \to (\Pi n{:}nat.\ (Pn \to P(succ\ n))) \to \Pi n{:}nat.\ Pn$

Computation rules:

$R\ nat\ P\ b\ f\ 0 \longrightarrow b$

$R\ nat\ P\ b\ f\ (succ\ n) \longrightarrow f\ n\ (R_{nat}\ P\ b\ f\ n)$

PRIMER: $P := \lambda\_{:}nat.\ nat \to nat$

$b := \lambda x{:}nat.\ x.$

$f := \lambda\_{:}nat.\ \lambda g{:}\ nat \to nat.\ \lambda y{:}nat.\ succ\ (g\ y)$

define $m+n := R_{nat}\ P\ b\ f\ m\ n$

real HW: compute $(succ\ 0) + (succ\ 0) \longrightarrow^* succ\ (succ\ 0)$

add (succ 0) (succ 0)
$\longrightarrow^*$ R_{nat} (λ_:nat. nat) (succ 0) (λ_:nat. λx:nat. succ x) (succ 0)
$\longrightarrow$ (λ_:nat. λx:nat. succ x) 0 (R_{nat} (λ_:nat. nat) (succ 0) (λ_:nat. λx:nat. succ x) 0)
$\longrightarrow^*$ succ (R_{nat} (λ_:nat. nat) (succ 0) (λ_:nat. λx:nat. succ x) 0)
$\longrightarrow^*$ succ (succ 0)

The type nat is an example of an inductive type.

Constructors: 0: nat suc: nat → nat

Eliminator: R_{nat} : ΠP : nat → type. P0 → (Πn. Pn → P (suc n)) → Π n:nat. Pn

computation rules: R_{nat} P b f 0 ⟶ b

R_{nat} P b f (suc n) ⟶ f n (R_{nat} P b f n)

Improved definition of addition:

Using R_{nat} we can define add: nat → nat → nat

add := λm:nat. λn:nat. R_{nat} (λ_:nat : nat) m (λ_:nat, x:nat. succ x) n

dva argumenta ločimo z vejico

β - reduction: $(\lambda x{:}A.\ t)\ s \to t[s/x]$

add (suc 0) (suc 0)

[(λm:nat, n:nat. R_{nat} (λ_:nat. nat) m (λ_:nat, x:nat. succ x) n)(suc 0)](suc 0) ⟶

(λn:nat. R_{nat} (λ_:nat. nat) (suc 0) (λ_:nat, x:nat. succ x) n)(suc 0) ⟶

R_{nat} (λ_:nat. nat)(suc 0) (λ_:nat, x:nat. suc x)(suc 0) ⟶

(λ_:nat, x:nat. suc x) 0 (R_{nat} (λ_:nat. nat) (suc 0) (λ_:nat, x:nat. suc x) 0) ⟶*

suc (R_{nat} (λ_:nat : nat) (suc 0) (λ_:int, x:nat. suc x) 0) ⟶ Suc (suc 0)

THE INDUCTIVE TYPE list A

Constructors: nil: list A , cons: A → list A ⟶ list A

Eliminator: R_{list} : ΠP : listA → Type. P nil → (Πa:A . Πas : listA . Pas → P(cons a as)) ⟶ Πas: listA . Pas

Computation rules: R_{list} P b f nil ⟶ b

R_{list} P b f (cons a as) ⟶ f a as (R_{list} P b f as)

HW: Use R_{list} to define append : list A → list A → list A

append [1 2] [3] ⟶* [1 2 3]

THE INDUCTIVE TYPE Vector A n

Constructors: nil: vector A 0 cons : $\Pi n : nat.\ A \to vector\ A\ n \to vector\ A\ (suc\ n)$

Eliminator: Rvector : $\Pi P : (\Pi n : nat.\ (vector\ A\ n \to Type)) \to$

$\to (\Pi n : nat.\ \Pi a : A.\ \Pi as : vector\ A\ n.\ P\ n\ as \to P\ (suc\ n)\ (cons\ n\ a\ as)) \to$

$\to \Pi n : nat.\ \Pi as : vector\ A\ n.\ P\ n\ as$

PROPOSITIONAL EQUALITY

Formation rule : $\dfrac{A : Type \quad s : A \quad t : A}{s =_A t : Type}$ proof relevant equality

$\dfrac{A : Type \quad s : A \quad t : A}{s =_A t : Prop}$ proof irelevant equality

Constructor: rfl : $\Pi t : A\ (t =_A t)$

Eliminator: J_A : $\Pi C : (\Pi x, y : A.\ (x =_A y) \to Type) \to$

$(\Pi x : A.\ C\ (x, x,\ refl)) \to$

$\Pi x, y : A.\ \Pi p : (x =_A y).\ C\ (x, y, p)$

Computation rule: $J_A\ c\ f\ t\ t\ rfl \to f\ t$

JUDGEMENTAL EQUALITY

$s \equiv t : A$ "s and t are equal by definition"

FORMATION RULE: $\dfrac{s : A \quad t : A \quad s \to^* u \quad t \to^* u}{s \equiv t : A}$

Substitution rule: $\dfrac{s \equiv t : A \qquad \boxed{\begin{array}{l} x : A \\ B : Type \end{array}} \qquad u : B[s/x]}{u : B[t/x]}$